Universität Regensburg

Wissenschaftliches Rechnen auf Grafikkarten

Übungsblatt

Thomas Karl

4. Juni 2020

Inhaltsverzeichnis

1 Bandbreitentest

Kopieren Sie ein einzelnes Array der Größe n in Bytes in den globalen Speicher. Messen Sie die Zeit t. Plotten Sie für verschiedene n die Zeiten und fitten sie linear die Funktion $t(n) = \frac{1}{b}n + l$ an. Dabei ist b die Bandbreite von PCIe (unidirektional) und l die Latenz.

2 Axpy

Parallelisieren Sie für zwei Vektoren $\vec{x}$ und $\vec{y}$ die Zuweisung

$$\vec{y} \leftarrow a \cdot \vec{x} + \vec{y} \tag{1}$$

auf der Grafikkarte. Schreiben Sie drei verschiedene Kernel für single-, double- und half-precision. Variieren Sie die Dimension der Vektoren und protokollieren Sie die Laufzeit der drei Versionen. Bestimmen Sie durch lineares Fitten den Speedup von half- und float- gegenüber double-precision.

3 Matrixaddition auf Cluster

Parallelisieren Sie eine einfache Addition zweier großer $n \times n$ Matrizen $C = A + B$ unter Verwendung mehrdimensionaler Blöcke. Benutzen Sie das Beispiel *cluster.cu* um das Problem auf verschiedene GPUs aufzuteilen. Überprüfen Sie das Gesamtresultat.

4 Bubble Sort

Implementieren Sie den *Bubble Sort Algorithmus*[1] sequentiell. Teilen Sie die innere Schleife auf: Die erste behandelt nur jedes zweite Element, die zweite die restlichen. Parallelisieren Sie diese nun unabhängigen Schleifen.

5 Concurrency

Messen Sie die Bandbreite wie in Augabe 1 für den bidirektionalen Fall. Kopieren Sie ein Array der Größe s auf die Grafikkarte. Benutzen Sie zwei asynchrone Kopierfunktionen um gleichzeitig dieses Array zu lesen und ein anderes der selben Größe zu schreiben. Dazu müssen diese Funktionen einem anderen Stream zugeordnet werden. Messen Sie die Zeit t für beide Operationen und fitten Sie wie in Aufgabe 1 mit $n = 2s$.

6 Streams

mehrere Additionen

Führen Sie mehrere Vektoradditionen auf dem selben Device aus. Maximieren Sie den Durchsatz, indem Sie jede Addition einem anderen Stream zuordnen und damit Concurrency ausnutzen.

eine Addition

Führen Sie eine Vektoradditionen auf dem selben Device aus. Maximieren Sie den Durchsatz, indem Sie die Arrays auf gleich große Subarrays aufteilen und jede Addition einem anderen Stream zuordnen. Führen Sie ihre Anwendung mit *Nvidia nsight* aus und überprüfen Sie mit dem Profiler den Überlapp.

7 Reduktionen

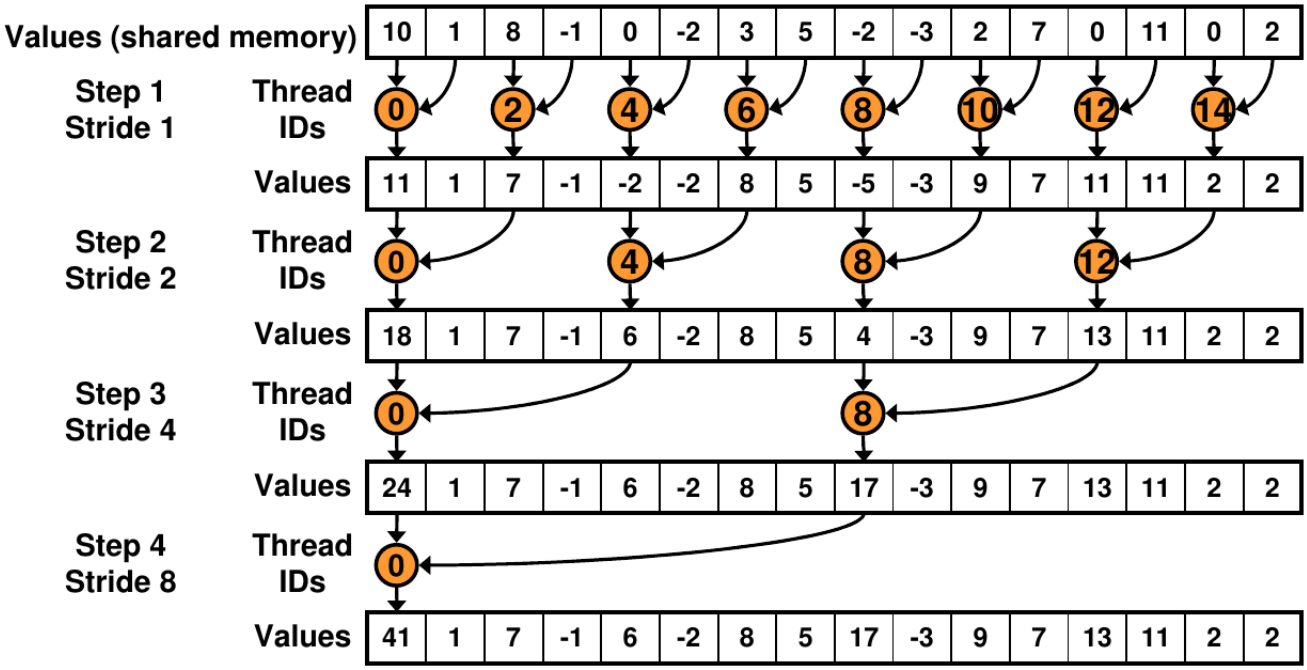
Schreiben Sie eine Reduktion in *CUDA*. Nehmen Sie an, Sie haben ein sehr großes Array mit Werten und wollen beispielsweise die Summe aller Elemente parallel berechnen. Nehmen Sie an, dass die Größe n des Arrays eine zweier-Potenz ist.

[1] https://de.wikipedia.org/wiki/Bubblesort

Schritt 1

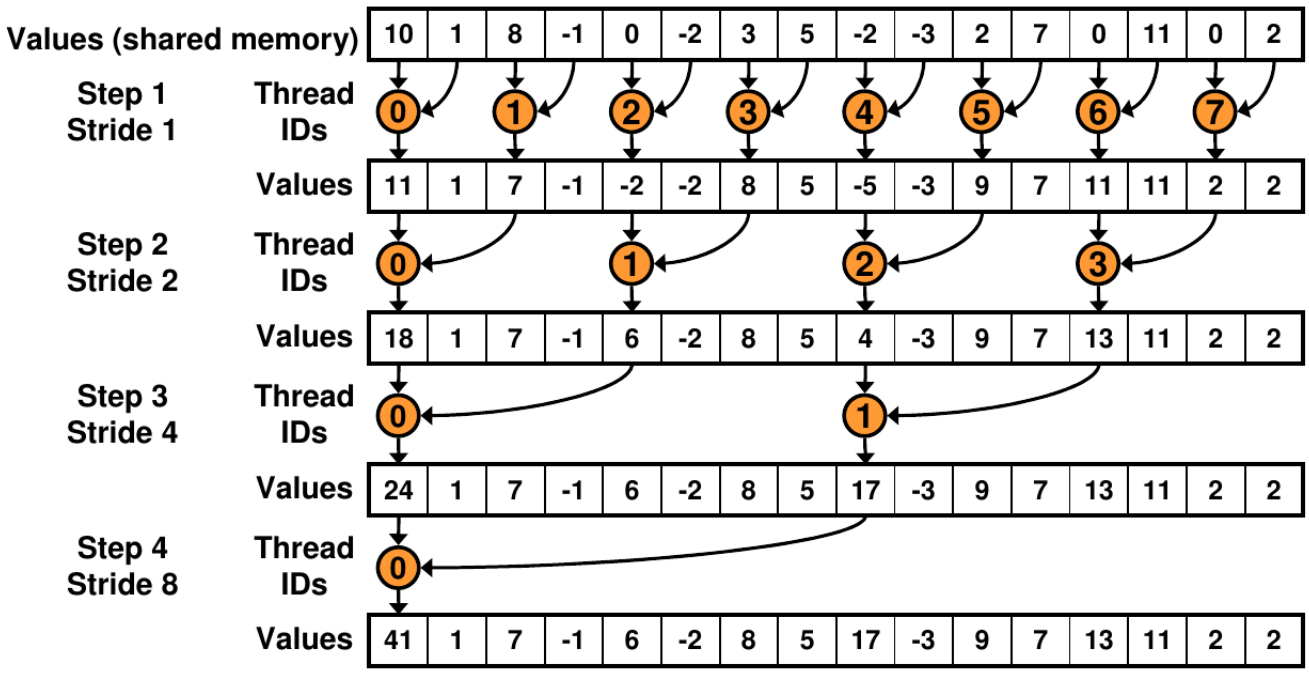
Vorgehensweise:

- Jeder Thread holt zwei Werte und speichert deren Summe im selben Array ab.
- Am zweiten Schritt ist nur noch die Hälfte der Threads beteiligt. Jeder Thread holt zwei der abgespeicherten Werte und berechnet erneut die Summe.
- Dies muss insgesamt $\log_2 n$ Mal ausgeführt werden.
- Die Summe wird also insgesamt zur ersten Position des Arrays durchgereicht.



Schritt 2

Vermeiden Sie Divergenz. Indizieren Sie die Threads so, dass alle beschäftigten Threads zusammenliegen.



Schritt 3

Laden Sie die Daten in den shared Memory. Vermeiden Sie dabei Bankkonflikte. Jeder Thread muss Werte lesen, die möglichst weit auseinander liegen.

Quelle: Mark Harris: Optimizing Parallel Reduction in CUDA
https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf

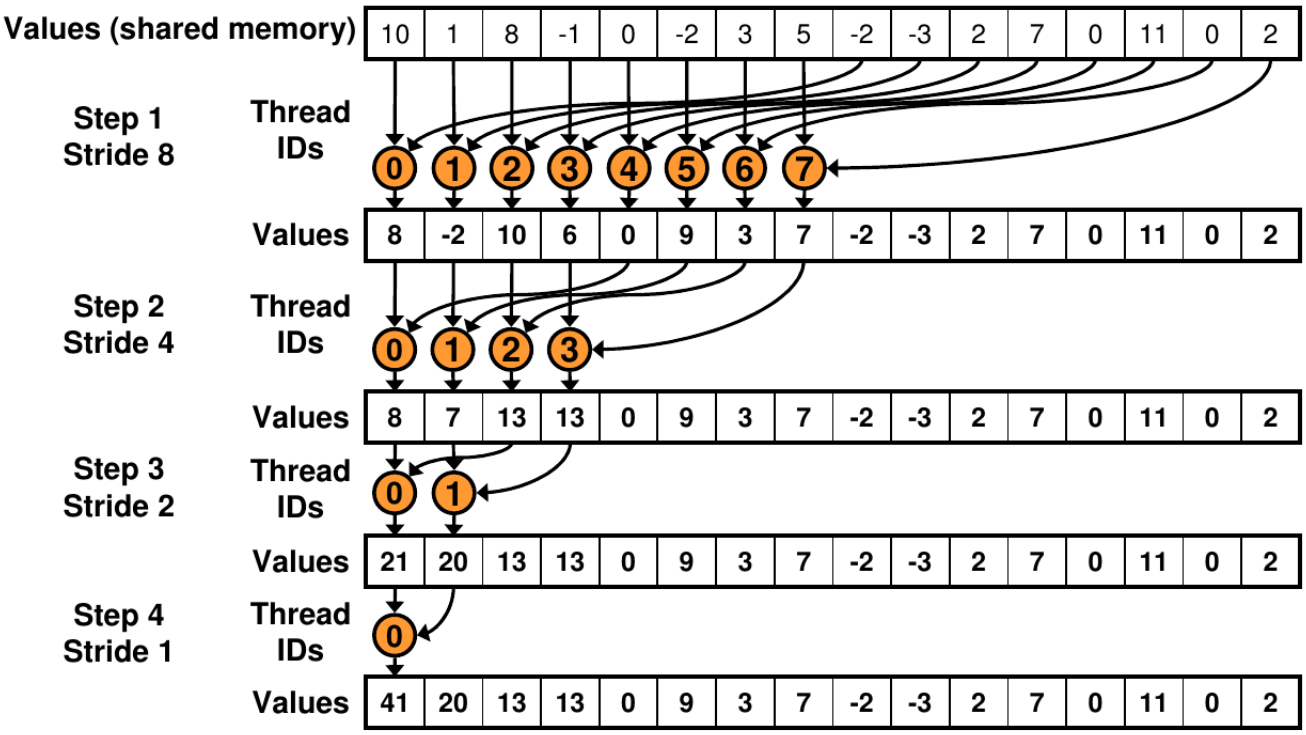
8 Molekulardynamik

Implementieren Sie den Verlet Algorithmus[2]: Es befinden sich n Teilchen in einem quadratischen Kasten mit Länge 1 und periodischen Randbedingungen. Jedes Teilchen wird repräsentiert durch den Ort $\vec{x}(t)$ zum Zeitpunkt t. Aus den Bewegungsgleichungen kann der Ort zu einem späteren Zeitpunkt $t + \Delta t$ berechnet werden:

$$\vec{x}(t + \Delta t) = 2\vec{x}(t) - \vec{x}(t - \Delta t) + \frac{\vec{f}}{m}\Delta t^2 + O(\Delta t^4) \tag{2}$$

- Vernachlässigen Sie zunächst die Kräfte.
- Würfeln Sie n 2d Ortsektoren $\vec{x}_0$. Berechnen Sie $\vec{x}_1 = \vec{x}_0 + \vec{v}t$ mit zufälligen Geschwindigkeiten.
- Führen Sie parallel auf der GPU für jedes Teilchen den Iterationsschritt $\vec{x}_2 = 2\vec{x}_1 - \vec{x}_0$ aus und speichern Sie $\vec{x}_2$ zwischen.

[2] https://en.wikipedia.org/wiki/Verlet_integration



- Kopieren Sie $\vec{x}_0$ nach $\vec{x}_1$ und $\vec{x}_2$ nach $\vec{x}_0$.
- Führen Sie das Kernel für jeden Zeitschritt aus, von $t = 0$ bis $t = t_{end}$ in Schritten von Δt.
- Berücksichtigen Sie nun die Kräfte: Die Iteration wird zu $\vec{x}_2 = 2\vec{x}_1 - \vec{x}_0 + \frac{\vec{f}_1}{m}\Delta t^2$ mit $\vec{x}_1 = \vec{x}_0 + \vec{v}t + \frac{\vec{f}_0}{m}\Delta t^2$.
- In jedem Schritt muss die Kraft $\vec{f}$ auf ein Teilchen i berechnet werden: $\vec{f} = \sum_j \text{const} \cdot \frac{\vec{x}^i - \vec{x}^j}{|\vec{x}^i - \vec{x}^j|^3}$ ($\vec{x}^i$ bezeichnet den Ort von Teilchen i).
- Wegen der Randbedingung sollten Sie nur die Kräfte durch Teilchen innerhalb eines bestimmten Cutoff-Radius berechnen.

9 Differentialgleichungen: Jacobi-Iteration

stationäre Wärmeleitungsgleichung in zwei Dimensionen

Mittels Finite Difference Methoden lassen sich DGLs oft gut auf der GPU parallelisieren. Dazu versucht man Ableitungsoperatoren über einen Differenzenquotient auf einem diskreten Gitter zu nähern. Die bekanntesten Differentialgleichungen ist die Wärmeleitungsgleichung

$$\frac{\partial}{\partial t}u(\vec{x},t) - a\,\triangle u(\vec{x},t) = f(\vec{x},t) \tag{3}$$

mit $a > 0$ für eine Temperatur u am Ort $\vec{x}$ zum Zeitpunkt t. Lösen Sie diese Gleichung im stationären ($\frac{\partial}{\partial t}u(\vec{x},t) = 0$) und homogenen ($f(\vec{x},t) = 0$) Fall. Gleichungen dieser Gestalt bezeichnet man als Laplace-Gleichungen. Definieren Sie ein quadratisches Gitter ($\vec{x} = (x, y)$) und belegen Sie die Randwerte mit festen, unveränderlichen Zahlen vor (Dirichlet-Randbedingung). Wählen Sie als Gitterkonstante in beide Raumrichtungen $h_x = h_y = 1$. Dann lässt sich die Laplace-Gleichung numerisch nähern als,

$$\triangle u(x, y) = u(x + 1, y) + u(x, y + 1) + u(x - 1, y) + u(x, y - 1) - 4u(x, y) = 0. \quad (4)$$

Also ist die Temperatur an jedem Ort der Mittelwert der vier Temperaturen der umgebenden Orte. Dieses Verfahren ist als *five-point-stencil* bekannt. Schreiben Sie eine Iteration, die in jedem Schritt für jeden inneren Ort (also außer am Rand) die Temperatur neu berechnet. Dazu müssen Sie die Schleife wieder in gerade und ungerade Punkte unterteilen, um unabhängige Loops zu erhalten. Überlegen Sie sich ein vernünftiges Konvergenzkriterium und überprüfen Sie das Ergebnis, indem Sie einige Zustände in einem farbigen Plot wie in Abbildung 1 darstellen[3].

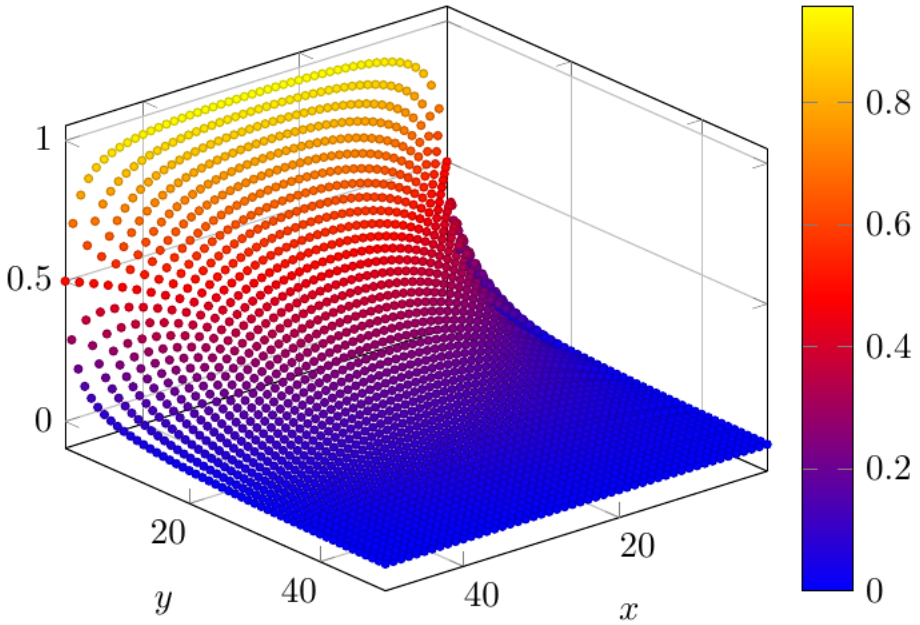


Abbildung 1: Wärmeleitende Platte mit konstanter Temperaturzufuhr am oberen Ende. In diesem Beispiel wurde die Gleichng (4) auf 50×50 Punkten diskretisiert. Die Randbedingung war $u(x, 0) = 1$ und $u(0, y) = u(50, y) = u(x, 50) = 0$ konstant.

zeitabhäbgige Wärmeleitungsgleichung in einer Dimension

Betrachten Sie einen wärmeleitenden Stab der Länge L, der durch die Funktion $u(x, t)$ beschrieben wird. Dies ist die Temperatur des Stabes an Ort $0 \leq x \leq L$ zum Zeitpunkt t. Die zu lösende

[3] Jeff Larkin: OpenACC Online Course 2018
https://www.openacc.org/sites/default/files/inline-files/OpenACC_Course_Oct2018/OpenACC%20Course%202018%20Week%201.pdf

homogene Differentialgleichung

$$\frac{\partial u}{\partial t} - \alpha \frac{\partial^2 u}{\partial x^2} = 0 \tag{5}$$

kann durch das Iterationsschema

$$u(x, t + \Delta t) = u(x, t) + \alpha \frac{\Delta t}{h^2}(u(x - h, t) - 2u(x, t) + u(x + h, t)) \tag{6}$$

mit h als Schrittweite der Diskretisierung genähert werden. Belegen Sie dazu $u(x, t = 0)$ beliebig vor und setzen Sie $u(0, t) = u(L, t) = 0$ als Randbedingung fest für jeden Zeitschritt. Plotten Sie die Temperaturverteilung am Anfang und am Ende, um ihr Ergebnis zu überprüfen.

10 THRUST

Saxpy

Schreiben Sie die *Saxpy*-Operation mittels *THRUST*-Funktoren, einmal mit Built-ins und einmal mit einem selbsgeschriebenen Funktor. Vergleichen Sie die Laufzeiten mit Aufgabe 2.

Varianz

Schreiben Sie einen Funktor zur Berechnung der Varianz eines Vektors.

Sortieren

Belegen Sie einen Devicevektor zufällig mit Ganzzahlen. Sortieren Sie die erste Hälfte absteigend nach dem Rest, den man erhält, wenn man die entsprechenede Zahl durch sieben teilt, die andere Hälfte aufsteigend.

11 cuRAND

Pseudorandom

Bestimmen Sie die Kreiszahl π durch Monte-Carlo Integration der Funktion $f(x) = \sqrt{1 - x^2}$ im Bereich $[0, 1]$. Würfeln Sie dazu im Quadrat $[0, 1) \times [0, 1)$ eine große Zahl von zweidimensionalen Punkten. Zählen Sie dabei mit, wie oft ein Punkt unterhalb der Funktion, also ob der Punkt auf

der Fläche unter dem Graphen liegt. Das Verhältnis dieser Zahl zur Gesamtanzahl sollte gegen die eingeschlossene Fläche konvergieren, also gegen $\pi/4$.

Quasirandom

Bestimmen Sie die Kreiszahl π durch Quasi-Monte-Carlo Integration. Nehmen Sie den Code aus der vorangegangenen Aufgabe und modifizieren Sie lediglich den RNG. Wahlweise lässt sich auch *clQMC* verwenden. Generalisieren Sie das Programm auf n-dimensionale Einheitskugeln und weisen Sie nach, dass deren Volumen für $n \to \infty$ gegen 0 konvergiert.

12 Fourier Transformation

Vollziehen Sie für ein beliebiges komplexwertiges Eingangssignal eine 3d *Complex-to-Complex* FFT in vier Versionen und vergleichen Sie die Laufzeit bei wachsender Signallänge:

- mit *cuFFT*
- mit *clFFT*
- mit *FFTW*
- mit dem *FFTW* Interface für *cuFFT*

13 BLAS

Vollziehen Sie in cuBLAS/clBLAS ein rank-1 update einer komplexwertigen hermiteschen $n \times n$ Matrix A,

$$A \leftarrow \alpha x x^H + A \tag{7}$$

für einen beliebigen Vektor x und ein Skalar α.

14 SPARSE

Implementieren Sie folgende Matrix mit cuSPARSE/clSPARSE:

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & & \cdots & 0 \\ -1 & 2 & -1 & 0 & & \vdots \\ 0 & -1 & 2 & -1 & 0 & \\ & \ddots & \ddots & \ddots & \ddots & \\ \vdots & & 0 & -1 & 2 & -1 \\ 0 & \cdots & & 0 & -1 & 2 \end{pmatrix} \tag{8}$$

Diese Matrix entspricht einem *three-point-stencil* und nähert Gleichung 5. Die Iteration 6 lässt sich auf den inneren Gitterpunkten schreiben als $\overline{u}_{t+\Delta t} = A\overline{u}_t$ mit $\overline{u}_t = (u(h,t), u(2h,t), ...u(L-h,T))^T$[4]. Vergleichen Sie die Lösung dieser direkten Methode mit ihrer selbstgeschriebenen Iteration oder mit Paralution.

15 Lineare Algebra

Vergleichen Sie drei Versionen einer Cholesky-Zerlegung der selben $n \times n$ Matrix A:

- (mit einer selbstgeschriebenen Version unter Zuhilfenahme von cuBLAS oder clBLAS)
- mit cuSOLVER
- mit MAGMA

A muss dazu symmetrisch und positiv-definit sein. Variieren Sie n und vergleichen Sie die Laufzeit. Testen Sie mit einer dünn besetzten Matrix und vergleichen Sie mit dem entsprechenden sparse-Algorithmus.

16 Projektideen

Velocity-Verlet Algorithmus

Erweitern Sie Aufgabe 8 zum Velocity-Verlet Algorithmus. Berechnen Sie dazu explizit die Geschwindigkeiten $\vec{v}(t+\Delta t) = \vec{v}(t) + \frac{1}{2m}(\vec{f}(t) + \vec{f}(t+\Delta t))\Delta t$. Berechnen Sie in jedem Schritt die Gesamtenergie als Summe der kinetischen Energien $\frac{1}{2}m\vec{v}^2$ von jedem Teilchen. Plotten Sie diese gegen die Zeitschritte, um so die Energieerhaltung zu überprüfen und einen Langzeitdrift der Energie

[4] Mehr Informationen unter http://www.cfd.tu-berlin.de/Lehre/tfd_skript/node46.html

auszuschließen.

Ransac Algorithmus

Der Ransac-Algorithmus („Random Sample Consensus“) ist ein simples Verfahren zum Fitten eines extrem verrauschten Signals. Simulieren Sie zunächst ein Signal von s Messpunkten, welches einer linearen Funktion folgt. Legen Sie ein Rauschen über dieses Signal, indem Sie zusätzlich im entsprechenden Werte- und Definitionsbereich eine extrem hohe ($>> 10^6$) Anzahl von uniform verteilten Punkten würfeln.

Der Algorithmus versucht nun eine optimale Gerade zu finden:

1. Wählen Sie zufällig zwei Punkte aus. Diese beiden Punkte legen eine Gerade eindeutig fest.

2. Bestimmen Sie die Anzahl an Punkten, die höchstens ε von der Gerade entfernt liegen. Übersteigt diese Anzahl einen bestimmten Wert w, wird diese Mengen zum sogenannten Consensus Set.

3. Wiederholen Sie die beiden Punkte beliebig oft. Das Ergebnis ist die Gerade, die den größten Konsens lieferte.

4. Die resultierende Menge kann nun für einen konventionellen Fit benutzt werden.

Es gibt zwei Möglichkeiten zur Parallelisierung.

- Parallelisieren Sie Punkt 3. Speichern Sie in jedem Schritt die Größe des Konsens. Vollziehen Sie anschließend eine Maximumsreduktion, um zu bestimmen, welche der Geraden am besten zu den Werten passt.

- Parallelisieren Sie Punkt 2, indem Sie die Menge aufteilen und zur Bestimmung des Konsens eine Summenreduktion vollziehen.

Die Parameter ε und w müssen a-priori bekannt sein und vom Nutzer eingegeben werden. ε sollte dabei in der Nähe der Varianz der Messwerte liegen, w etwa bei deren Anzahl. Dazu muss also vorher (falls möglich) das Signal untersucht oder die Werte durch trial-and-error bestimmt werden. w kann in jedem Schritt angepasst werden. Ein neuer Konsens wird nur akzeptiert, wenn dieser eine Verbesserung des alten darstellt.

Die Anzahl an benötigten Iterationen m lässt sich durch Methoden der Stochastik abschätzen. Soll mit einer Wahrscheinlichkeit von p mindestens einmal ein Konsens auftreten, so gilt für diese Anzahl $m = \frac{\log(1-p)}{log(1-(1-(s-w)/s)^s)}$. Besteht ein Signal zu 70% aus Rauschen, so sind als nur 49 Iterationen nötig, um mit einer Wahrscheinlichkeit von 99% mindestens einmal einen Konsens zu erhalten.

Ising Modell

Replica Exchange

Nullstellen komplexer Polynome

G-DBSCAN

Random Forest